EECE 4811/5811: HW6 Answers

Problem 1: Latency Numbers

Compare Jeff Dean's 2009 latency numbers to current values (2024).

Below is a table showing the comparison and sources.

Prompt Used:

"Compare Jeff Dean's 2009 latency numbers to current values (2024). Include updated values for: L1/L2 cache, memory, disk seek, SSD read, network latency, etc. Provide units in nanoseconds and cite your sources."

Operation	2009 Latency (ns)	2024 Latency (ns)	Source
L1 cache reference	**0.5**	**~0.5**	**https://erickguan.me/2024/latency-v**
Branch mispredict	**5**	**~8**	**https://erickguan.me/2024/latency-v**
L2 cache reference	**7**	**~2.5**	**https://erickguan.me/2024/latency-v**
Mutex lock/unlock	**25**	**~25**	**https://erickguan.me/2024/latency-v**
Main memory reference	**100**	**~70**	**https://erickguan.me/2024/latency-v**
Read 1MB from memory	**250,000**	**~10,000**	**https://erickguan.me/2024/latency-v**
Datacenter round trip	**500,000**	**~100,000**	**https://erickguan.me/2024/latency-v**
Disk seek (HDD)	**10,000,000**	**10,000,000**	**https://louwrentius.com/understand**
Read 1MB from SSD	**20,000,000**	**~100,000**	**https://erickguan.me/2024/latency-v**
Send packet CANLCA	**150,000,000**	**~150,000,000**	**https://erickguan.me/2024/latency-v**

Problem 2: Process and Page (Copy-On-Write)

Q1: What is Copy-On-Write (COW) and Why It Helps

Copy-On-Write (COW) is an optimization used during fork() where the parent and child processes initially

share the same memory pages marked as read-only. Only when either process attempts to write to a page does the OS create a private copy (i.e., on write). This reduces memory use and speeds up the fork operation, especially when large memory spaces are involved or when exec() is called soon after fork().

Q2: Pseudo-code for Fork with COW

```
function fork_with_cow(parent_process):
    child_process = create_new_process()
    for each page in parent_process.page_table:
        mark page as read-only
        share page with child
        increment ref_count
        map page into child_process.page_table
    set_page_fault_handler(child_process, handle_cow_page_fault)
    return child_process

function handle_cow_page_fault(process, page):
    if page is shared and read-only:
        if ref_count[page] > 1:
            new_page = allocate_new_page()
            copy contents from page to new_page
            map new_page as writable
            decrement ref_count[page]
        else:
            mark page as writable

function terminate_process(process):
    for each page in process.page_table:
        decrement ref_count[page]
        if ref_count[page] == 0:
            free_physical_page(page)
```